

Atelier Data Science

Seminar 1

Neural Net Foundations

Irina Proskurina

Irina.Proskurina@univ-lyon2.fr

Laboratoire ERIC – Université Lyon 2

Course Plan

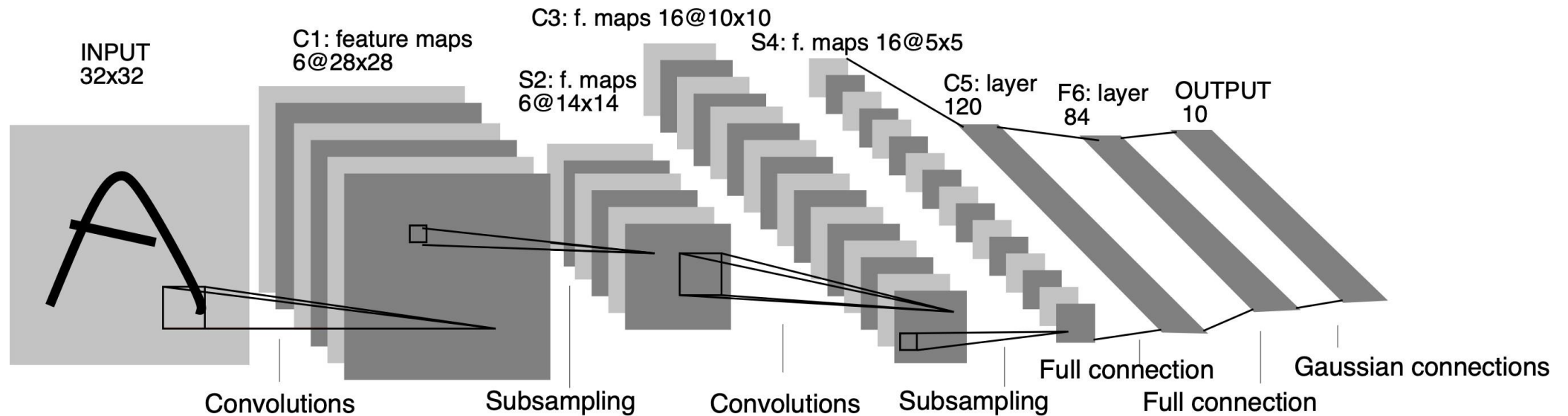
- Neural Net Foundations (Recall)
- From Model to Production
- Convolutions (CNNs)
- Optimization
- Random forests and model interpretation
- Word2Vec
- Attention&Transformers

Link : <https://github.com/upunaproks/ul2-atelier-data-science>

Traditional Computer Vision

1. Feature engineering (whiskers, the shape of the ears, the length of the tail, etc.)
 2. Train gradient boosting on features
- Feature engineering – too difficult and costly

Modern Deep Computer Vision



Traditional NLP

1. Count the frequency with which a specific word appears following another ones
2. Generate the next word based on this distribution

"Manure, almond gelato and frozen pies, you are also had it was in one but it will post office buildings sucks). their chinese food. comfort food while they liked their lids ripped off. it an early morning of jon still a spade so maybe too much. the same. but, at the baked rigatoni, and not in other options and it seems odd taste). our visit). i go to nfl kickoff arrived with \$. that's about when you come down hoyt street is actually higher than impressed with a regular theater! so at it, halfway through their pork and though i've"

Modern NLP

1. GPT-3 — a neural network trained on a massive corpus of texts.

The article you are writing about is going to be based around this new technology, so you have been spending a lot of time playing around with it. You have also been using your own brain to test out the new models, which is something no one else in the world has done. As a result, you have become somewhat obsessed with it. You constantly think about how it can create such fantastic sentences and how it might be used to solve the world's problems.

Recent achievements

- Images and videos
- Three-dimensional computer vision
- Texts
- Sound
- Data generation

Useful Links

- <https://www.deeplearningbook.org>
- <https://cs231n.github.io/convolutional-networks/>
- <https://course.fast.ai>

Why do we need neural networks?

Predicting the cost of an apartment

- Linear Model:

$$a(x) = w_0 + w_1 * (\text{area}) + w_2 * (\text{floor}) + w_3 * (\text{location}) + \dots$$

- These features are likely not independent of each other

Predicting the cost of an apartment

- Linear model with polynomial features:

$$\begin{aligned} a(x) = & w_0 + w_1 * (\text{area}) + w_2 * (\text{floor}) + w_3 * (\text{location}) \\ & w_4 * (\text{area})^2 + w_5 * (\text{floor})^2 + w_6 * (\text{location})^2 \\ & w_7 * (\text{area}) * (\text{floor}) \\ & + \dots \end{aligned}$$

- How to interpret this model?
- What is $(\text{area}) * (\text{floor})$?

Gradient boosting

$$a_N(x) = \sum_{n=1}^N b_n(x)$$

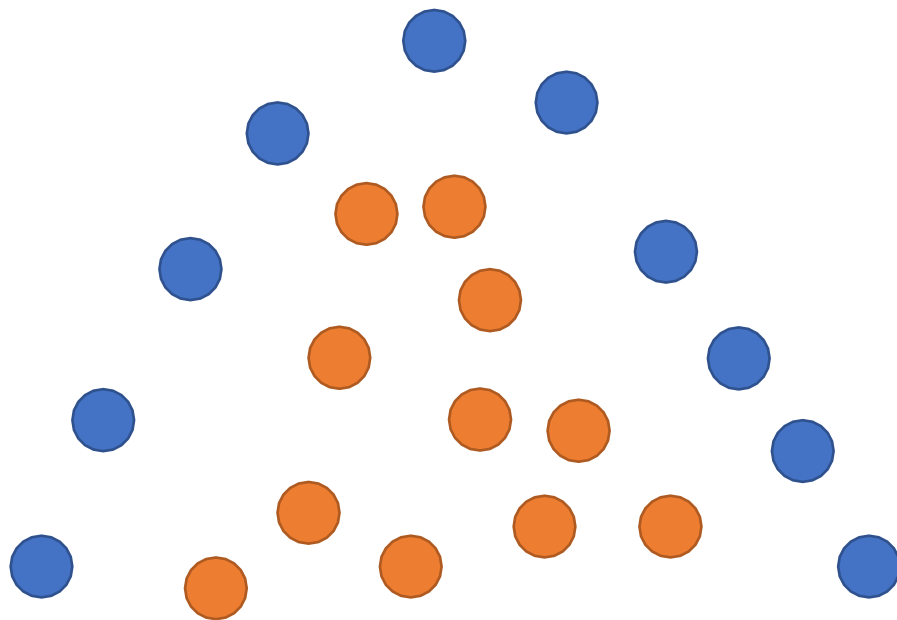
The N -th model:

$$\frac{1}{\ell} \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + b_N(x_i)) \rightarrow \min_{b_N(x)}$$

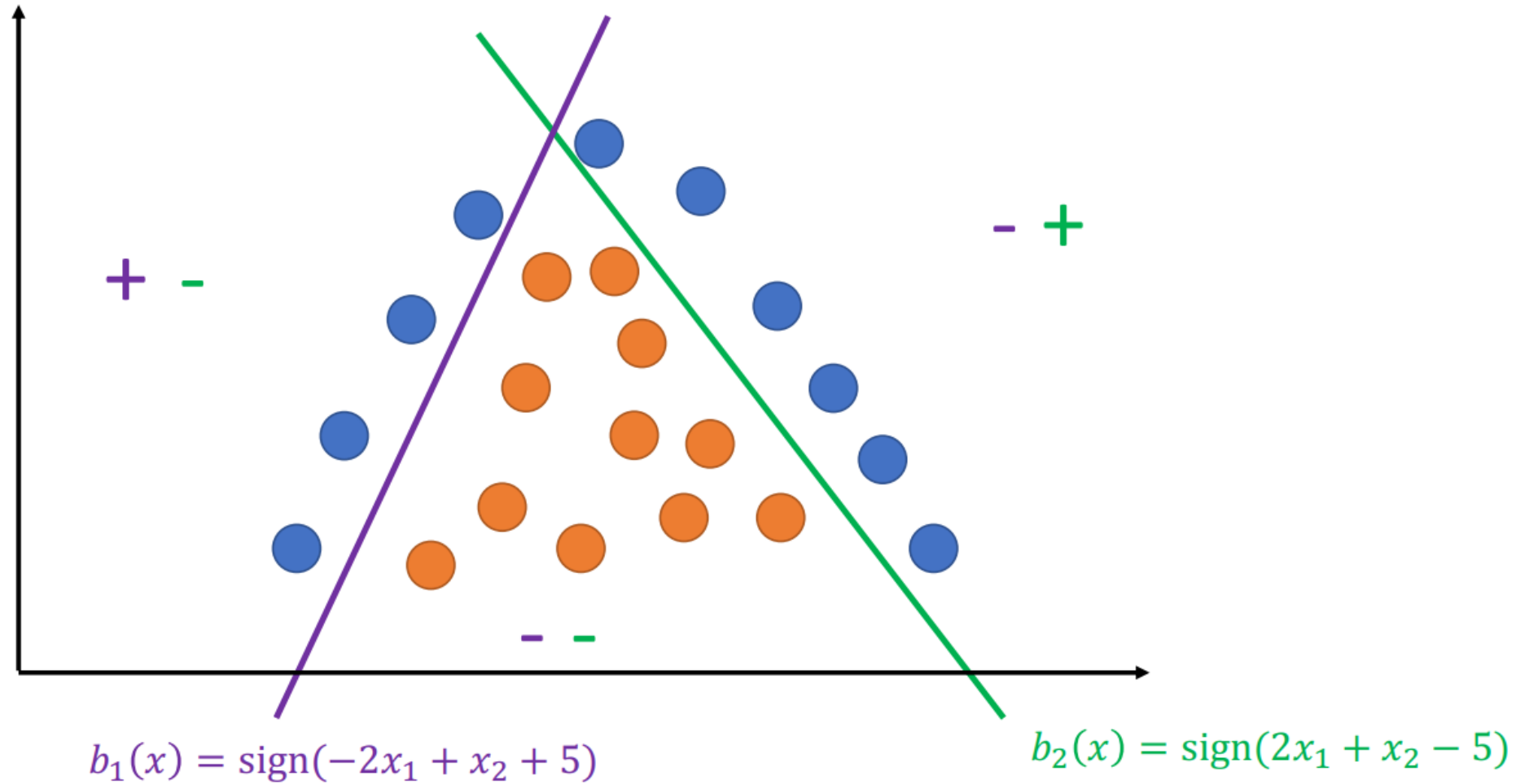
Summary

- Linear models can be trained using gradient descent but are not well-suited for capturing complex patterns
- Decision trees and their ensembles provide better results but are challenging to train

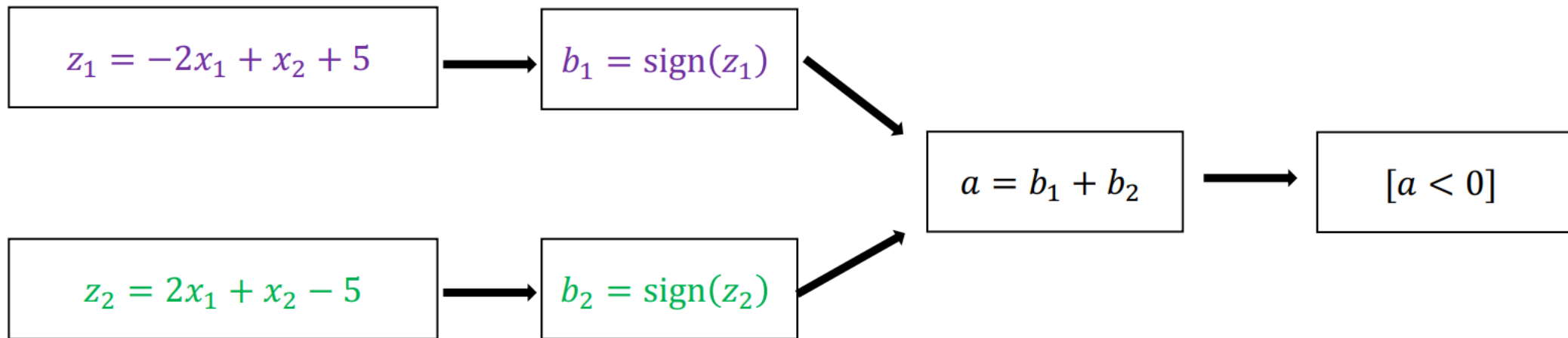
Nonlinear patterns



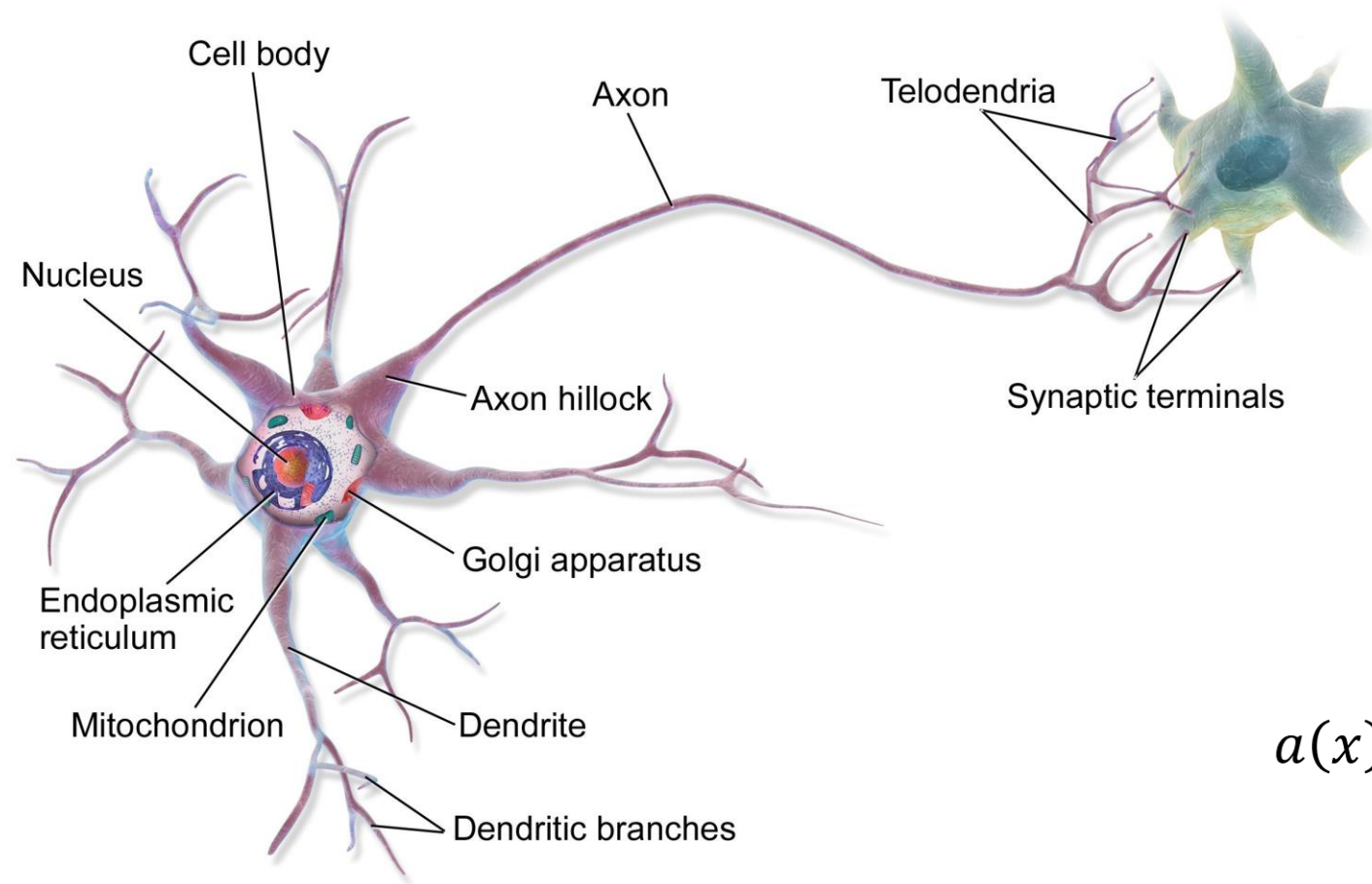
Nonlinear patterns



Nonlinear patterns

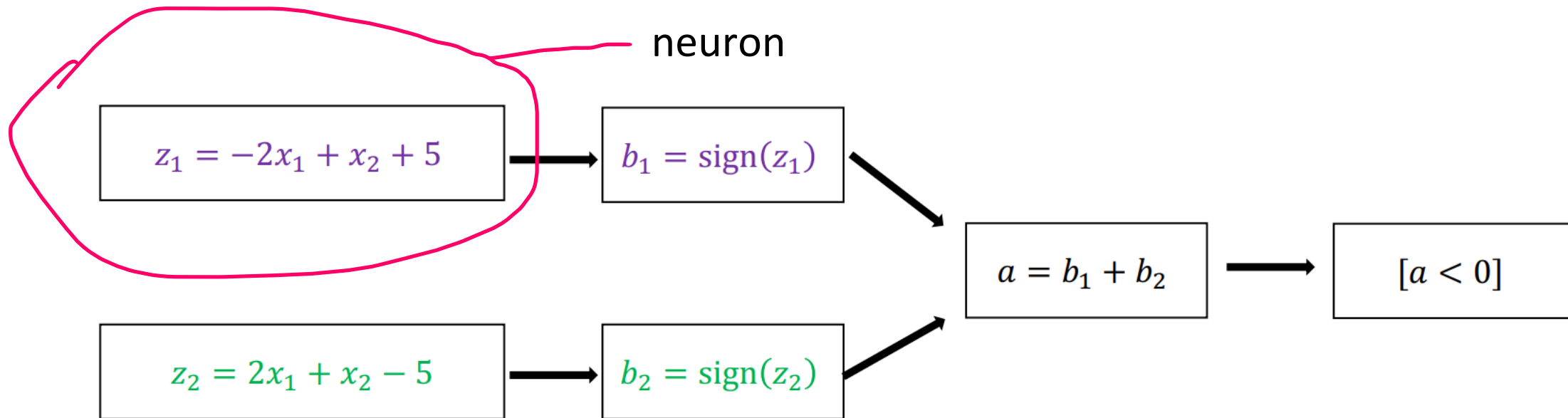


Neuron



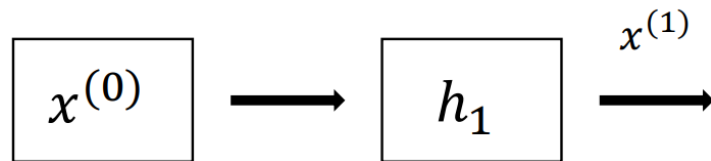
$$a(x) = \sum_{j=1}^d w_j x_j$$

Nonlinear patterns

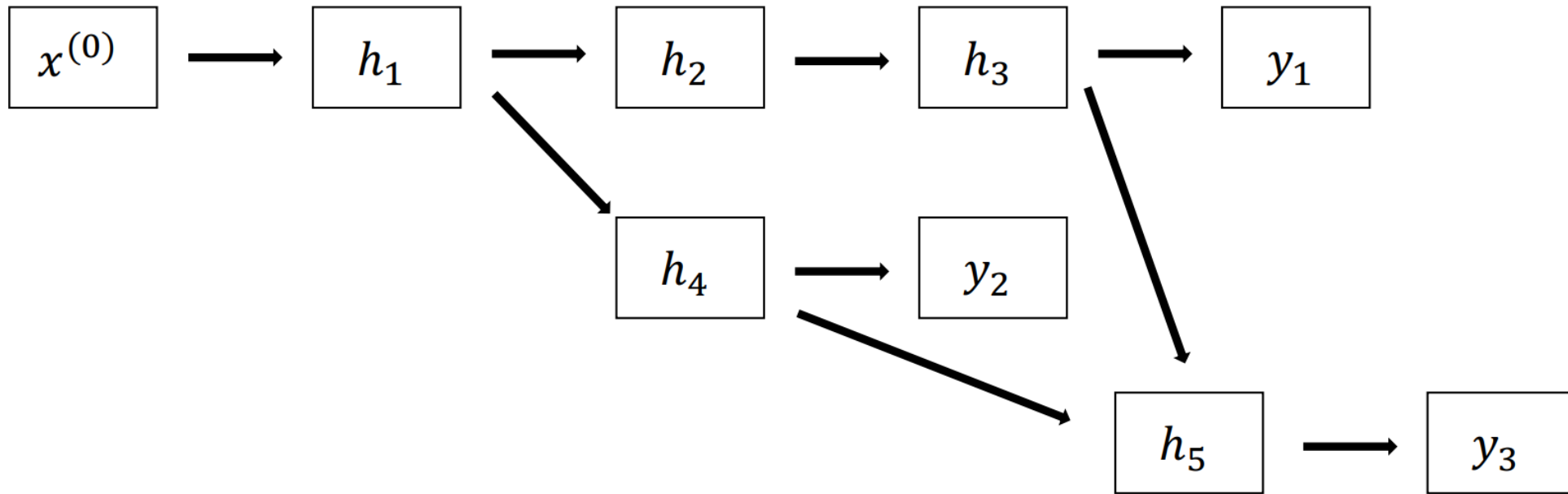


Computational graph (neural network)

- $x^{(0)}$ — input features
- $h_1(x)$ — layer
- $x^{(1)}$ — output



Computational graph (neural network)



Fully connected layers

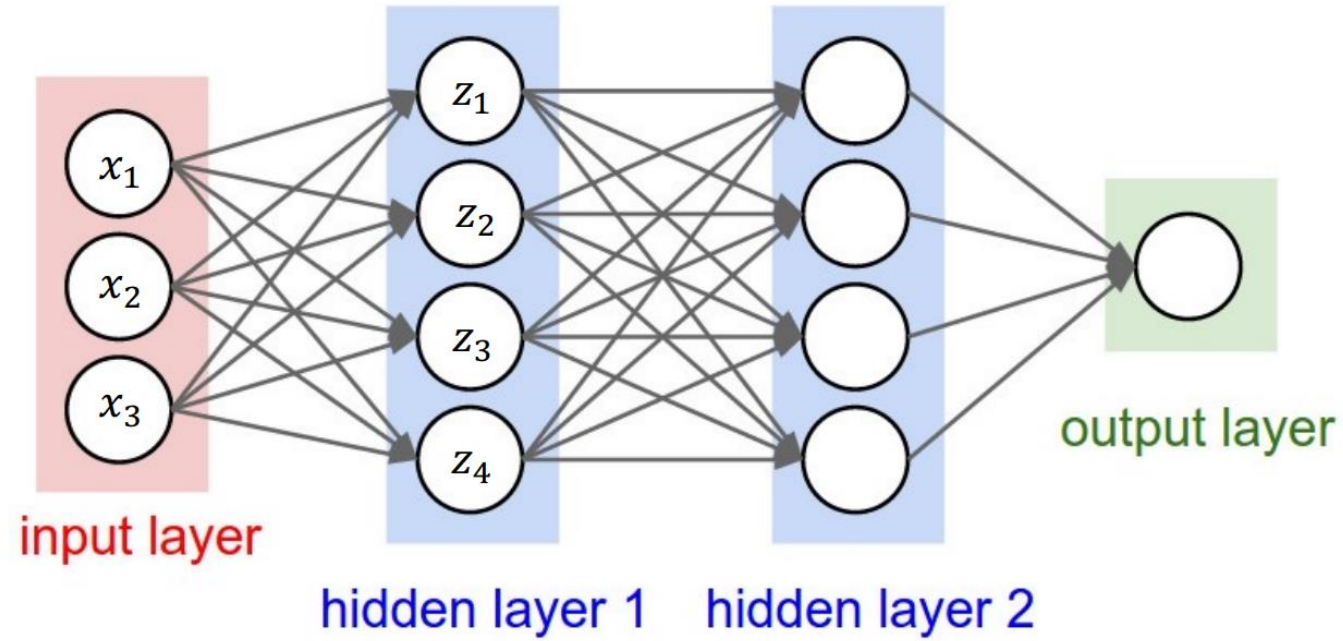
Fully connected layers

- The input consists of N values, and the output produces M values
- x_1, \dots, x_N — input
- y_1, \dots, y_M — output
- Each output is the result of applying a linear model to the inputs.

$$z_j = \sum_{i=1}^n w_{ji} x_i + b_j$$

Fully connected layers

$$z_j = \sum_{i=1} w_{ij} x_i + b_j$$



Fully connected layers

$$z_j = \sum_{i=1}^n w_{ji} x_i + b_j$$

- m linear models, each with $(n + 1)$ parameters
- in total, there are approximately mn parameters in a fully connected layer

Fully connected layers

$$z_j = \sum_{i=1}^n w_{ji} x_i + b_j$$

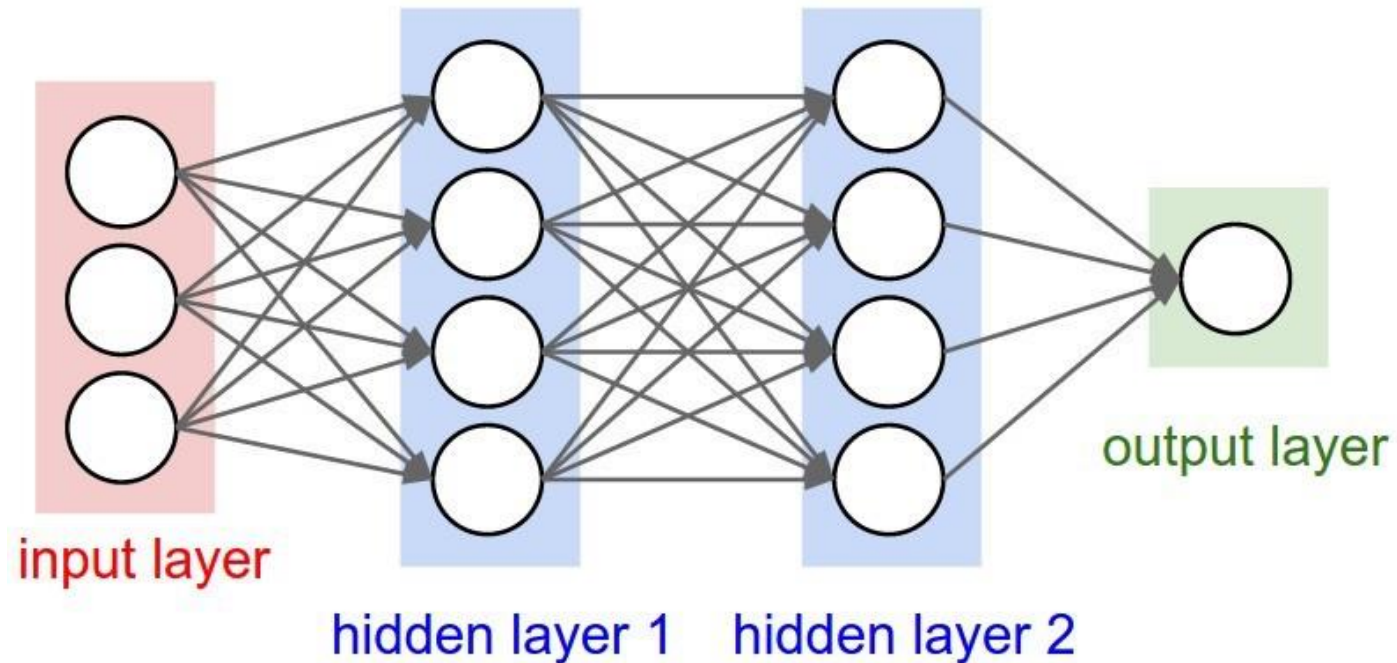
- m linear models, each with $(n + 1)$ parameters
- in total, there are approximately mn parameters in a fully connected layer
- if we have 1,000,000 input features and 1000 outputs, it amounts to 1,000,000,000 parameters
- a substantial amount of data is needed for training

Important Questions in DL

How to build a useful model in deep learning?
Which layers to add?

Non-linearity

- Given 2 fully-connected layers



Non-linearity

- Given 2 fully-connected layers

$$\begin{aligned} s_k &= \sum_{j=1}^m v_{kj} z_j + c_k = \sum_{j=1}^m v_{kj} \sum_{i=1}^n w_{ji} x_i + \sum_{j=1}^m v_{kj} b_j + c_k = \\ &= \sum_{j=1}^m \left(\sum_{i=1}^n v_{kj} w_{ji} x_i + v_{kj} b_j + \frac{1}{m} c_k \right) \end{aligned}$$

- So, this is no better than a single fully connected layer

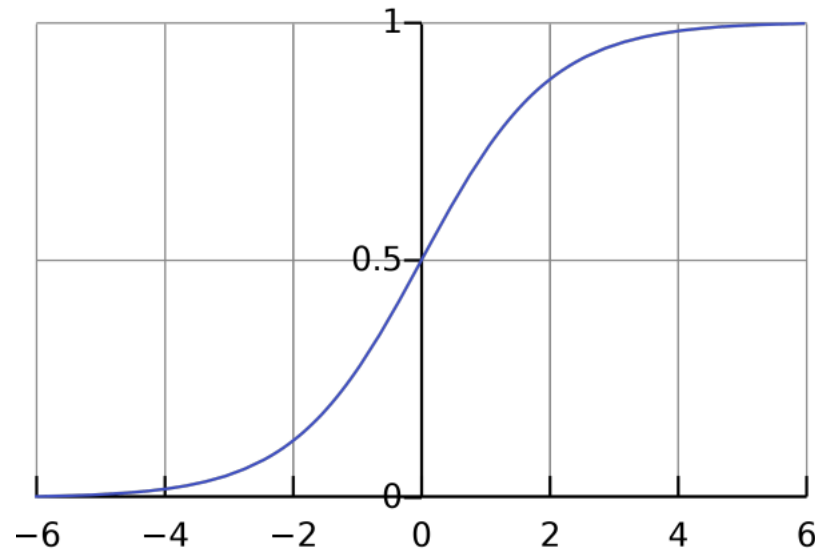
Activation functions

- It is necessary to add a non-linear activation function after the fully connected layer

$$z_j = f\left(\sum_{i=1}^n w_{ji}x_i + b_j\right)$$

1. $f(x) = \frac{1}{1+\exp(-x)}$

Logistic/sigmoid



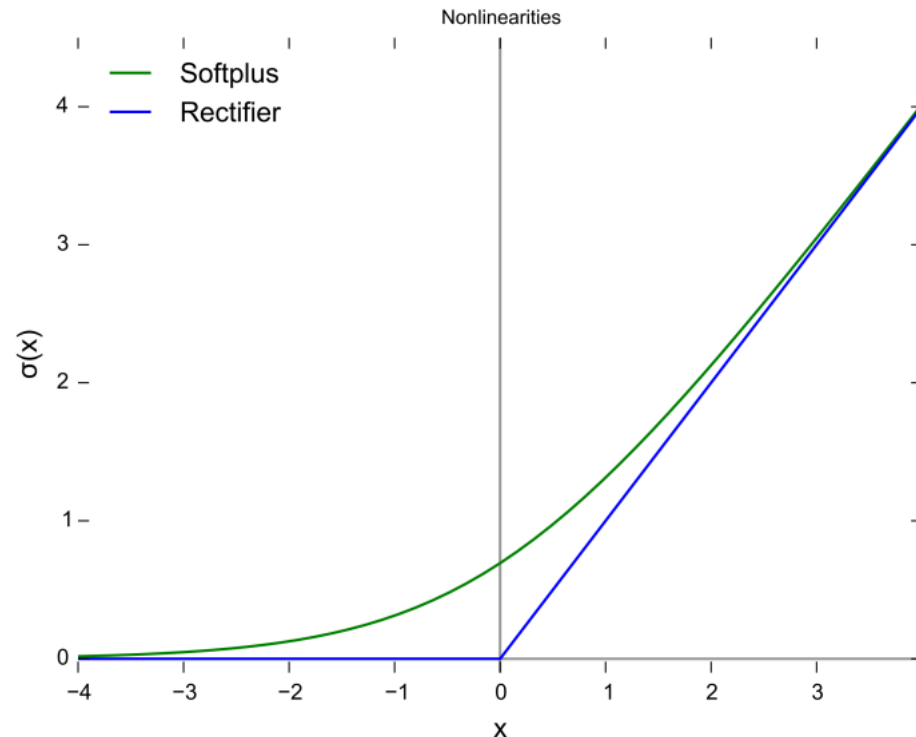
Activation functions

- It is necessary to add a non-linear activation function after the fully connected layer

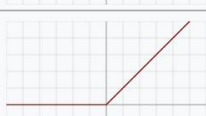
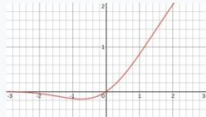
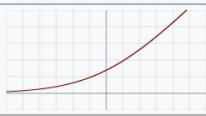

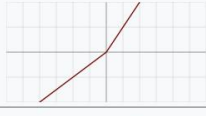
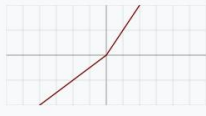
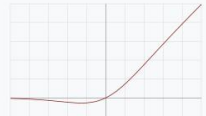
$$z_j = f\left(\sum_{i=1}^n w_{ji}x_i + b_j\right)$$

2. $f(x) = \max(0, x)$

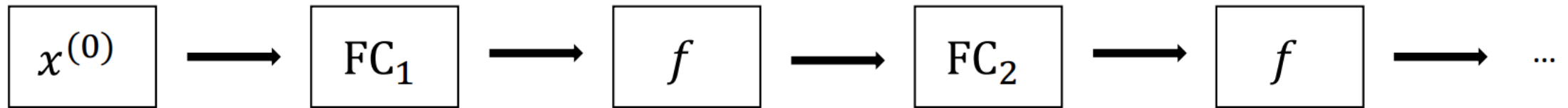
(ReLU, REctified Linear Unit)



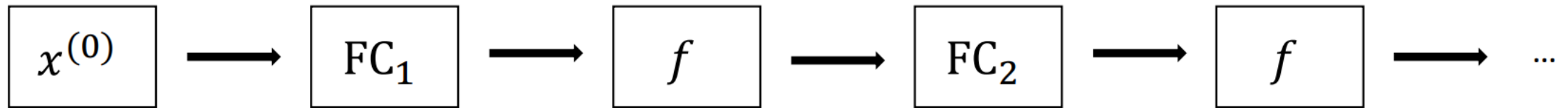
Activation Functions

Rectified linear unit (ReLU) ^[9]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x \mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU) ^[4]		$\frac{1}{2}x \left(1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$ $= x\Phi(x)$
Softplus ^[10]		$\ln(1 + e^x)$
Exponential linear unit (ELU) ^[11]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ <p>with parameter α</p>
Scaled exponential linear unit (SELU) ^[12]		$\lambda \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ <p>with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$</p>
Leaky rectified linear unit (Leaky ReLU) ^[13]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
Parameteric rectified linear unit (PReLU) ^[14]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ <p>with parameter α</p>
Sigmoid linear unit (SiLU, ^[4] Sigmoid shrinkage, ^[15] SiL, ^[16] or Swish-1 ^[17])		$\frac{x}{1 + e^{-x}}$

A fully connected neural network



A fully connected neural network



- Features are fed into the network
- The dimension of the last layer = # target variables

The Cybenko Universal Approximation Theorem

Summary:

Let $g(x)$ be a continuous function. Then, it is possible to construct a two-layer neural network that approximates $g(x)$ with any predefined precision.

In other words, two-layer neural networks are VERY powerful!